BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

LINEAR LOGISTIC REGRESSION DEMO

DEMONSTRAČNÍ APLIKACE LINEÁRNÍ LOGISTICKÉ REGRESE

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **ADAM BAK**
AUTOR PRÁCE

SUPERVISOR **Ing. KAREL BENEŠ**
VEDOUCÍ PRÁCE

BRNO 2018

Zadání bakalářské práce/20291/2017/xbakad00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel:	**Bak Adam**
Obor:	Informační technologie
Téma:	**Demonstrační aplikace lineární logistické regrese**
	Linear Logistic Regression Demo
Kategorie:	Zpracování řeči a přirozeného jazyka

Pokyny:

1. Seznamte se s lineární logistickou regresí jako modelem pro pravděpodobnostní klasifikaci
2. Navrhněte demonstrační aplikaci, která bude zobrazovat různé aspekty trénování logistické regrese
3. Implementujte navrženou aplikaci
4. Vytvořte plakát shrnující výsledky práce

Literatura:

- C. M. Bishop: Pattern Recognition and Machine Learning

Pro udělení zápočtu za první semestr je požadováno:

- Body 1, 2 a rozpracování bodu 3.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí:	**Beneš Karel, Ing.**, UPGM FIT VUT
Datum zadání:	1. listopadu 2017
Datum odevzdání:	16. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

LS

[illegible]

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstract

This bachelor's thesis deals with the machine learning model logistic regression. The aim is to closely inspect and analyze the workings of this model for classification in order to be able to provide a learning tool in the form of demonstrative application. All of the mathematical formulae, logistic sigmoid, cross entropy error function and gradient are derived and explained in detail. This thesis also provides some insight into the form of the cross entropy error function in the case of linear logistic regression.

Abstrakt

Táto bakalárska práca sa zaoberá lineárnou logistickou regresiou, modelom pre strojové učenie. Cieľom tejto práce je podrobne preskúmať a zanalyzovať ako tento klasifikačný model funguje, aby bolo možné vyvinúť učebnú pomôcku vo forme demonštračnej aplikácie. Všetky matematické rovnice, logistická sigmoida, chybová funkcia vzájomnej entropie, metóda najväčšieho spádu sú odvodené a podrobne vysvetlené. Táto práca tiež prináša náhľad do tvaru grafu chybovej funkcie vzájomnej entropie v prípade lineárnej logistickej regresie.

Keywords

machine learning, logistic regression, classification, generalized linear models, probabilistic generative models, logistic sigmoid, cross entropy error function, gradient descent

Kľúčové slová

strojové učenie, logistická regresia, klasifikácia, generalizované lineárne modely, pravdepodobnostno generatívne modely, logistická sigmoida, chybová funkcia vzájomnej entropie, metóda najväčšieho spádu

Reference

BAK, Adam. *Linear Logistic Regression Demo.* Brno, 2018. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Karel Beneš

Rozšírený abstrakt

Táto bakalárska práca je zameraná na model lineárnej logistickej regresie, ktorý sa v strojovom učení používa na klasifikáciu. Zámerom tejto práce je podrobne preskúmať a zdokumentovať fungovanie tohto modelu. S pomocou získaných poznatkov potom navrhnúť a implementovať demonštračnú aplikáciu, ktorá môže byť použitá ako náučná pomôcka pre prehĺbenie vedomostí z problematiky klasifikácie a modelov strojového učenia.

V úvodnej časti práce, v kapitole 1 je zdôvodnená motivácia výberu témy práce strojového učenia a klasifikácie, popisuje tiež predmet a ciele tejto práce.

V nasledujúcej kapitole 2 je rozpracovaný úvod do problematiky klasifikácie. Sú tu popísané druhy klasifikačných problémov spolu s konkrétnymi príkladmi pre jednotlivé klasifikačné problémy. Ďalej sa v nej spomína roľa pravdepodobnosti a neurčitosti v klasifikácii. Sú tu zmienené marginálne, združené a podmienené pravdepodobnosti spolu so spôsobom ako ich vypočítať. Hovorí sa tu o vzťahoch medzi týmito pravdepodobnosťami, a je tu v krátkosti spomenutá Bayesova veta. Tieto pravdepodobnosti sa v neskorších kapitolách používajú pri prispôsobovaní modelu. V tejto kapitole sa spomína aj spôsob rozhodovania pri klasifikácii, rôzne možnosti nad ktorými sa musí človek zamyslieť pri riešení takýchto problémov. Je tu aj porovnanie prístupu troch rôznych typov modelov strojového učenia. Sú to modely s diskriminačnou funkciou, pravdepodobnostno generatívne modely a pravdepodobnostno diskriminatívne modely. Po nich nasledujú lineárne modely pre klasifikáciu a ich rozhodovacie hranice. Ďalej je tu vysvetlená lineárna regresia a roľa, ktorú hraje v lineárnych modeloch pre klasifikáciu. Na záver kapitoly sú uvedené aktivačné funkcie a ako s ich pomocou je možné previesť kvantitatívnu premennú na kvalitatívnu.

V nasledujúcej kapitole 3 sa rozoberá lineárna logistická regresia ako klasifikačný model. Je tu odvodené a popísané fungovanie tohto modelu spolu s dôvodmi prečo je ho vhodné používať. Rozoberá sa tu spôsob modelovania posteriórnych pravdepodobností a ich vzťah so vzorcom pre nepriazeň. Aktivačná funkcia logistická sigmoida, ktorá mapuje celú osu y z lineárnej kombinácie parametrov na interval $(0, 1)$. Kritériá použité pri trénovaní modelu, ciže chybová funkcia vzájomnej entropie. A nakoniec samostatné trénovanie modelu pomocou metódy najväčšieho spádu. Všetky vzorce sú dôkladne sformulované a ich odvodenie je popísané do detailu. Na konci tejto kapitoly by malo byť jasné fungovanie modelu lineárnej logistickej regresie a spôsobu akým ju možno použiť na klasifikáciu prostredníctvom posteriórnych pravdepodobností, ktoré modeluje.

V kapitole 4 je spísaný návrh a implementácia demonštračnej aplikácie pre lineárnu logistickú regresiu. Sú tu uvedené jednotlivé dôležité prvky tohto modelu, ktoré je potrebné sprístupniť užívateľovi. Užívateľ potrebuje byť schopný vytvárať trénovacie súbory dát, ktoré si bude môcť uložiť a načítať. Na týchto súboroch dát potom bude trénovať klasifikačný model. Je potrebné zobraziť detailné informácie o procese trénovania. To znamená, že po každom kroku algorimu sa obnovia informácie o parametroch modelu, chybovej funkcii a posteriórnych pravdepodobnostiach pre jednotlivé body z trénovacieho súboru. Ďalej bolo nutné na grafe ukázať vzťah medzi hodnotou lineárnej kombinácie parametrov a hodnotou funkcie logistickej sigmoidy. Ako poslednú vec bolo treba znázorniť priebeh chybovej funkcie vzájomnej entropie. Je to funkcia troch parametrov, čiže je nutné zobraziť 4-rozmerné dáta. To bolo realizované pomocou 3-rozmerného bodového grafu s farbou pre hodnotu chybovej funkcie v danom bode.

V predposlednej kapitole 5 sú zdokumentované poznatky, ktoré som zistil o priebehu chybovej funkcie vzájomnej entropie. Je to konvexná funkcia, ktorá pre veľmi vhodné riešenia klasifikačného problému má hodnotu blízku nule. Je zaujímavé, ako je táto funkcia často symetricky rozčlenená do regiónov v priestore. Je to z dôvodu, že pre ľubovoľnú

hodnotu chybovej funkcie je možné nájsť nekonečne veľa rovníc priamok, ktoré budú na seba lineárne závisle. Rovnako pre ľubovoľne dobré riešenie klasifikačného problému existuje takisto zlé riešenie, ktoré ma posteriórne pravdepodobnosti obrátené.

Na záver práce je v kapitole 6 stručne zhrnutý obsah práce, postup pri jej vypracovaní a dosiahnuté výsledky. Sú tu taktiež navrhnuté spôsoby, ako v práci naďalej pokračovať a rôzne možnosti pre rozšírenia.

Linear Logistic Regression Demo

Declaration

Hereby I declare that this bachelor's thesis was prepared as an original author's work under the supervision of Ing. Karel Beneš. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

. . . .
Adam Bak
May 16, 2018

Acknowledgements

I would like the thank Ing. Karel Beneš for his supervision and indispensable help in writing this thesis. Further, I would like to thank my family for their unending support without whom this would not have been possible.

Contents

Chapter 1

Introduction

Machine learning is a field of computer science that has been around for decades. It has been gaining a lot of traction in the past couple of years, mainly due to the immense amount of data generated that requires processing. In 2013 the amount of data on the Internet was around 4.4 zettabytes. This amount is expected to grow exponentially and is estimated to reach 44 zettabytes in 2020 [15].

Machine learning is an inter-disciplinary field which brings together computer science and statistics. With use of machine learning the immense amount of data that is available to humankind can be processed more efficiently. It finds application in text classification, object recognition, speech recognition, behavioral prediction, etc. Machine learning permeates our society so completely that every person most likely uses some aspects of it daily without even knowing.

This emergent popularity of machine learning has generated a lot of interest, even in people who are not too familiar with this field. A lot of the study materials available fall into one of the two categories. They either oversimplify the subject to be understandable by the average layperson, or assume a preexisting knowledge of the matter at hand. This represents a barrier to entry, which creates a need for focused literature that covers chosen topics in their entirety while being very clear, concise and understandable at the same time. So much so, that a person could use these texts as the entry point for a deeper understanding of the problems of classification and the machine learning models.

The aims of this thesis are to inspect and analyze the workings of classification in general and logistic regression model in particular, to be able to provide a concise explanation of all the mathematics and algorithms behind the model. The goal is to create a complementary learning tool to go along with the documentation of classification and logistic regression. This demonstrative application provides a hands-on experience to anyone looking to more closely understand this particular machine learning model. The demo application is be able to display the most important parts of the logistic regression model as well as show the user how different parameters and setting of the model impact the training process and the outcome of the classification.

The linear logistic regression model for classification was chosen due to its explainability. It is not too difficult to understand the relationship between the input and the output in this model. Logistic regression is one of the more popular algorithms used for classification. The main advantage it offers is the fact that it models the conditional probabilities. This is in practice a lot more useful than strictly classifying data. Logistic regression models these conditional probabilities directly, avoiding the computational complexity of models that infer conditional probabilities from prior class probabilities.

The following Chapter 2 discusses the classification in general. It serves as a form of introduction into given problematic. This Chapter mentions what constitutes the classification problem, shortly explains the probability theory and the different probabilities that are talked about in this thesis. It delves into the problems of optimal decision making and decision theory. And lastly, it deals with the subset of classification models called linear models.

In Chapter 3 the working of logistic regression are explained in depth. This Chapter contains the derivations of all the mathematical formulas used in this machine learning models. The process of training the model, as well as the relationship between the inputs and the outputs are shown in this Chapter.

Chapter 4 documents the design and implementation of the demonstrative application. The software and libraries that were used in the process of creating the application are mentioned there.

Chapter 5 documents the findings about the form of the cross entropy error function and its plot. Namely the symmetry of the functions plot and the regions of what would be good solutions for the model where the posterior probabilities are inverted.

The last Chapter 6 contains a summary of the work that was put into this thesis, the outcomes and some prospects for the future of this work that might warrant further looking into.

Chapter 2

Classification

The following chapter covers the basic theory behind classification. It focuses mainly on the underlying principles involved in the process, such as uncertainty and forming optimal decisions.

Classification is a problem of trying to correctly assign an object, or an event, to one of N discrete *classes* C_n. A set of preselected features used to decide the outcome of classification is called an *input vector* $\vec{x}$. It is a set of measured values used to calculate the value of the *response variable* y. The response variable y can be subsequently used to decide on the class assignment of the object represented by $\vec{x}$. The outcome of classification is a label t, which represents one of the classes. Label is a categorical variable from a finite set of $t \in \{C_1, C_2, ..., C_n\}$.

The value of the response variable y is determined by an adaptive model. The parameters of this model are altered throughout the training process to achieve a more optimal outcome of classification. It is repeatedly adjusted based on a *training set* of data. The training set consists of a number of pairs $(\vec{x}_i, t_i)$, where $\vec{x}_i$ is D-dimensional vector of discrete or continuous values and t_i is its corresponding label.

To give an example of a classification problem, consider trying to distinguish between two types of fruit, apples and oranges. The input vector could consist of three features: weight, color and smoothness. If provided with a set of data about 1000 apples and oranges, it would be possible to train a model that would distinguish between them. This model might learn to associate heavier, orange colored, bumpy objects with oranges and lighter, red or green, smooth objects with apples.

2.1 Types of Classification Problems

Classification problems can be differentiated based on two factors, number of classes N and the number of labels L the algorithm can output.

Binary classification is the case when $N = 2$ and $L = 1$. In this case classifier outputs a single label corresponding to one of the two classes. It is often assumed that the values for C_1, C_2 are 0 and 1 respectively. An example of a binary classification problem is the apple-orange classification described above.

Whenever $N > 2$, we are talking about *multi-class classification*. The number of outputs for any given data is still $L = 1$. This is the same problem as before, however now it would include additional types of fruit like grapefruit, mango etc. A way of solving this type of problem is training multiple models. There are two approaches to this.

First is the one-vs-all approach, where a classifier is constructed for each of the classes. These classifiers are then fitted against the rest of the classes. The output label is then decided based on which of the models is the most confident. The second is the one-vs-one approach, where a classifier is constructed for each pair of classes. There are therefore multiple classifiers for each class. The outcome is decided by summing the confidence of these classifiers for each of the classes.

Lastly, when the number of output labels can be $L > 1$, we are talking about *multi-label classification*. In these cases a single input vector can belong to multiple classes. Trying to decide what types of fruit are displayed in a picture, if multiple different kinds appear simultaneously, is an example of multi-label classification.

The binary classification is the most convenient for demonstrating how classification works, therefore it is the primary focus in this thesis. It is also possible to adapt binary classification to solve multi-class and problems by using multiple binary classifiers.

(a) $y \in \{\textbf{Yes}, \textit{No}\}$ (b) $y \in \{\textit{Apple}, \textbf{Orange}, \textit{Pear}, \textit{Lemon}\}$ (c) $y \subset \{\textbf{Apple}, \textbf{Orange}, \textit{Pear}, \textit{Lemon}\}$

Figure 2.1: Binary, multi-class and multi-label classification problems, where the following questions are asked. (a) Does the image contain fruit? (b) Which fruit is in the image? (c) Which types of fruit are in the image?

The Figure 2.1 illustrates the difference between these three types of classification problems. In each example the same picture is used and the only thing changing are the questions and the set of possible answers.

The case (a) is a binary classification problem. The outcome is a simple yes or no depending on whether the picture contains a fruit.

In the case of (b) the image contains two types of fruit in the picture. However, the output can be only one of the two fruits that are in the picture, as it is a multi-class classification. The oranges are more numerous and take up more space in the picture, therefore the output is orange.

Lastly, in (c), the outcome is both apples and oranges. It is a multi-label classification problem and it is possible to choose more than one answer, if multiple types of fruit are represented in the image.

2.2 Probability Theory in Classification

Whether it be through the act of measuring the features of observed object, or the fact that the training datasets are not infinite, there is always a uncertainty that arises in machine learning and classification in particular. All of the observed data are most likely natural occurring events, which are bound to have outliers.

The probability theory is a core concept in machine learning. It is used to quantify this uncertainty and error. There are two notable views of what probability is, *frequentist interpretation* and *Bayesian interpretation* [9].

The frequentist interpretation views probabilities of events as the frequency of their occurrence given large enough sample scale. It posits that the probability is equal to the limit of observed relative frequency of occurrence, as the number of trials approaches infinity.

$$p(x) = \lim_{n \to \infty} \frac{n_x}{n} \tag{2.1}$$

The Bayesian interpretation is, that probabilities provide a quantification of uncertainty [1]. If we take a coin toss as an example, Bayesian interpretation would view the probability of 0.5 as the next time a coin is tossed, there is a 50% chance it will land tails side up. This is fundamentally different from the previous approach where we would expect that given N trials the frequency of the coin landing tail side up would approach $N/2$.

With Bayesian interpretation of probability, it is possible to consider uncertain events that will not occur repeatedly. With polar ice caps melting quite rapidly, it is possible that they will disappear entirely. This is an event that has never been observed before. However, with enough information about the rates at which ice melts, it would be possible to predict probability of their disappearance by year 2020, 2030 etc.

In machine learning, the goal is often to find the likelihood of something, such as the likelihood of an object belonging to a certain class. The probability theory expresses the likelihood of something, given our assumptions. It can be used when training the model and making optimal decisions.

2.2.1 Joint, Marginal and Conditional Probabilities

There are different types of probabilities that are commonly talked about in probability theory. To explain these probabilities, take two variables X and Y. Variable X can have values x_i, where $i \in \{0, 1, ..., N\}$ and variable Y can have values y_j, where $j \in \{0, 1, ..., M\}$. Three of these probabilities that are important for this thesis are:

1. Joint probability of $X = x_i$ and $Y = y_j$, is the probability that X has value x_i and Y has value y_j. It can be written as $p(X = x_i, Y = y_j)$. It is the probability of both events x_i and y_j happening together. Joint probability has the symmetry property, meaning that $p(X = x_i, Y = y_j) = p(Y = y_j, X = x_i)$. There is a way to calculate it using the *product rule*, as a product of posterior probability $p(Y = y_j|X = x_i)$ and marginal probability $p(X = x_i)$.

$$p(X = x_i, Y = y_j) = p(Y = y_j|X = x_i)p(X = x_i) \tag{2.2}$$

2. Marginal probability of $p(X = x_i)$, is the probability that X has the value x_i regardless of the value of Y. It can be obtained by summing all of the joint probabilities where x_i is one of the pair.

$$p(X = x_i) = \sum_{j=0}^{M} p(X = x_i, Y = i_j) \tag{2.3}$$

This is often referred to as the *sum rule*. It is called the marginal probability, because it is obtained by marginalizing, or summing out, the other variables (in this case Y) [1].

3. Conditional probability of $p(X = x_i|Y = y_j)$, is the probability that the X takes value x_i given that $Y = y_j$. Bayes theorem defines a relationship between conditional probabilities as follows

$$p(Y = y_j|X = x_i) = \frac{p(X = x_i|Y = y_j)p(Y = y_j)}{p(X = x_i)}. \tag{2.4}$$

This is derived from the symmetry of joint probability and the product rule.

Although in the above definitions the most rigorous notation is used, it is also rather lengthy. From now on instead of $p(X = x_i)$ we use the notation $p(X)$ to denote a distribution over the random variable X, or $p(x_i)$ to denote the distribution evaluated for the particular value x_i [1]. With this in mind, the joint probability can be written as $p(X, Y)$, marginal probability as $p(X)$ and conditional probability as $p(X|Y)$.

	t = 0	t = 1
x = 3	$\frac{2}{5}$	0
x = 4	$\frac{1}{5}$	$\frac{2}{5}$

(a) Joint probability p(x,t)

	t = 0	t = 1
x = 3	1	0
x = 4	$\frac{1}{3}$	$\frac{2}{3}$

(b) Posterior probability p(t|x)

Table 2.1: The difference between probabilities modeled in generative and discriminative models. Shown on dataset $\{(3,0),(3,0),(4,0),(4,1),(4,1)\}$ consisting of 5 ordered pairs (x,t).

To better understand the difference between joint and posterior probabilities, take a look at the table 2.1. On this small sample size, the posterior probability of $p(t = 0|x = 3) = 1$. That is because, in all cases when $x = 3$, the $t = 0$ as well.

However, the joint probability of $p(t = 0, x = 3)$ is only equal to $\frac{2}{5}$, as x and t only take these values twice, out of five data points provided in this dataset.

2.3 Decision Theory and Optimal Decision-Making

The problem of classification is trying to decide on correct class label to attach to a set of measured features. Decision theory, when combined with probability theory, allows for optimal decision making in situations involving uncertainty such as those encountered in pattern recognition [1]. Classification problems can, for the most part, be broken down into two parts - inference and decision.

To make optimal decision it is first necessary to obtain information about current problem by training a model. Take for example input vector $\vec{x}$ and a set of labels $t \in \{t_0, t_1, ..., t_n\}$. A way to approach this problem could be trying to learn the joint probability of $p(x, t)$, based on the available training data, and then make predictions about the value of t for given $\vec{x}$. The first part of this problem is called inference, it is often far the more difficult of the two. Once that is done, making the optimal decision for given case is usually very easy.

To illustrate what the decision making process can entail, look at the following problem. This is an example that is often given in these circumstances, as it shows the different impact a misclassification can have. Consider a problem, where the goal is to diagnose a patient that has potentially contracted a deadly disease. The outcome of the classification will be $t \in \{0, 1\}$, representing classes C_0 and C_1 respectively. If $t = 0$ it will mean that this patient does not have the disease and $t = 1$ will mean that he has the disease. The goal is to predict this value of t based on the results of a number of conducted tests and correctly assess the patients state. The results of these tests will be the input vector $\vec{x}$. The machine learning algorithm that is used will output the conditional probability of $p(C_0|\vec{x})$ and $p(C_1|\vec{x})$, where $p(C_0|\vec{x}) = 1 - p(C_1|\vec{x})$.

The outcome of the classification is decided based on the higher of the two values. Something to consider in this situation is what would happen if model is uncertain about the class membership. What if the probability that $\vec{x}$ belongs to class C_0 is 90% . This would still leave a 1 in 10 chance that the patient has the deadly disease. In cases like this it would be best to decide what is an acceptable margin of error. Then depending on the result, perhaps more closely inspect the situation at hand. For example run more tests or have a human expert handle the situation. This is also known as the reject option.

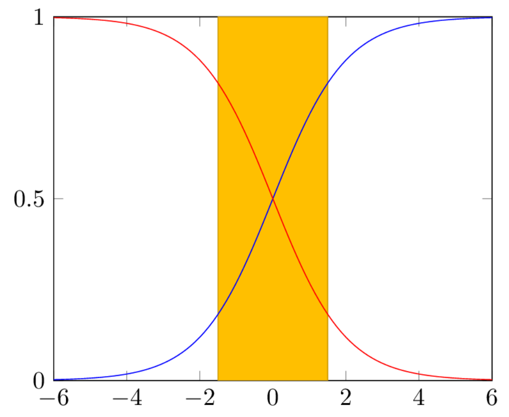


Figure 2.2: The reject option, in the highlighted region the decisions made by the algorithm should be inspected again due to unacceptably high possibility of error.

Another aspect of this problem is that of false positives. How do we deal with the marginal cases, when the result was misclassified despite our best efforts. This is a valid question because of the different impact our decision has depending on what happened. There are four possible outcomes in our problem $t = 1$ and the person is sick, $t = 1$ and the person is healthy, $t = 0$ and the person is healthy and $t = 0$ and the person is sick. The two cases when our model is in error are $t = 1$ and the person is healthy and $t = 0$ and the person is sick. Although the former is a misclassification, its impact is relatively minor as the person does not in fact have the disease. The latter, however, is a grave error on the part of our model and this might result in death of the patient. It would be possible to solve this issue during the training of the model, by penalizing it more heavily whenever such an error occurs.

	t = 0	t = 1
Healthy	0	1
Sick	1000	0

Table 2.2: Penalty incurred in the learning process.

The penalty incurred could look something like the example provided in Table 2.2. This way the model eventually learns to err on the side of caution in cases, where in the past it has misclassified sick patients as healthy.

2.3.1 Forming Decision

There are three distinct ways to approach solving classification problems.

1. First of them is to use what is called a *discriminant function* $f(\vec{x})$. This function maps every input vector $\vec{x}$ in the input space to one of N classes C_n. This approach combines the inference and decision into one. Solving the problem by finding a linear combination of features for which the misclassification is minimal.

2. Second approach is the one used in probabilistic generative models. Here the aim is to solve the inference problem of determining the class-conditional densities $p(\vec{x}|C_n)$ for each class C_n individually [1]. Also separately infer the prior class probabilities $p(C_n)$. Then use Bayes' theorem in the form

$$p(C_n|\vec{x}) = \frac{p(\vec{x}|C_n)p(C_n)}{p(\vec{x})}. \tag{2.5}$$

As stated in Subsection 2.2.1, we can find $p(\vec{x})$ by first calculating the joint probability $p(C_n, \vec{x}) = p(\vec{x}|C_n)p(C_n)$ and next marginalizing to get marginal probability

$$p(\vec{x}) = \sum_{n=0}^{N} p(C_n, \vec{x}). \tag{2.6}$$

However, it is also possible to leave out $p(\vec{x})$, as it only acts as a sort of normalizer. It ensures that the resulting posterior probability is from a range of $0 \leq p(C_n|\vec{x}) \leq 1$.

With that we have found the posterior probability $p(C_n|\vec{x})$, which can be used to assign new input vector $\vec{x}$ into one of the classes C_n.

Equally valid approach is to model the joint probability $p(C_n, \vec{x})$ directly and again by marginalizing obtain marginal probability $p(\vec{x})$. Thus finding the posterior probability.

3. The last approach is that of probabilistic discriminative models. These models try to model the posterior probability of $p(C_n|\vec{x})$ directly and again using this probability to determine assignment to classes C_n for given input vectors $\vec{x}$.

Using discriminant functions is the simplest and most straightforward of the three options. However, combining both of the inference and decision making into one we lose the ability to determine posterior probabilities $p(C_n|\vec{x})$, which are useful for things other than class assignment. For example we would lose the ability to apply the reject option from Section 2.3.

The most complex of these approaches is the technique used in generative models. It may require more training data and computational time in order to accurately model the class-conditional likelihoods $p(x|C_n)$. An advantage these models have is that they have access to the marginal probabilities $p(\vec{x})$ for the input data. These can be used to determine when an input vector is an outlier for which the model can make less accurate predictions.

However, sometimes determining the class-conditional probabilities and joint probabilities might prove unnecessarily difficult when all we want is to classify data. For which, all we need are the posterior probabilities $p(C_n|x)$ modeled directly by discriminative models.

2.3.2 Minimizing the Rate of Misclassification

When trying to optimize how well the model is doing in classifying new data, it is first necessary to be able to measure its performance. This is done by trying to optimize an objective function, which is integral to machine learning. This is a general term used for a function that is being optimized, it might also be called utility function. It is the function, which is being either maximized or minimized, to achieve best results and optimal decision making when assigning labels.

Formally speaking, there are two approaches to improving the performance of a model. It is possible to either maximize the likelihood of classifying correctly or minimize the rate of misclassification. Both of these approaches give equivalent results, but in the case of most algorithms it is customary to minimize misclassification. A function that measures the cost associated with an action or decision is called a *cost function*. In the case of classification problems it measures the expected loss, or rate of misclassification, for given model on the training data. It can be thought of as the overall performance of a model given available data.

There is not a clear why most of the machine learning models try to minimize a cost function instead of maximizing likelihood. One of the possible explanations is that, in numerical analysis, the vast majority of scientific papers focus on convex optimization problems and minority on concave optimization. And since cost functions are convex for simple linear models, the machine learning field has followed suit.

That does not mean that all of the algorithms follow this convention. For example maximum likelihood estimate (MLE) and maximum a posteriori estimation (MAP) both maximize their objective function.

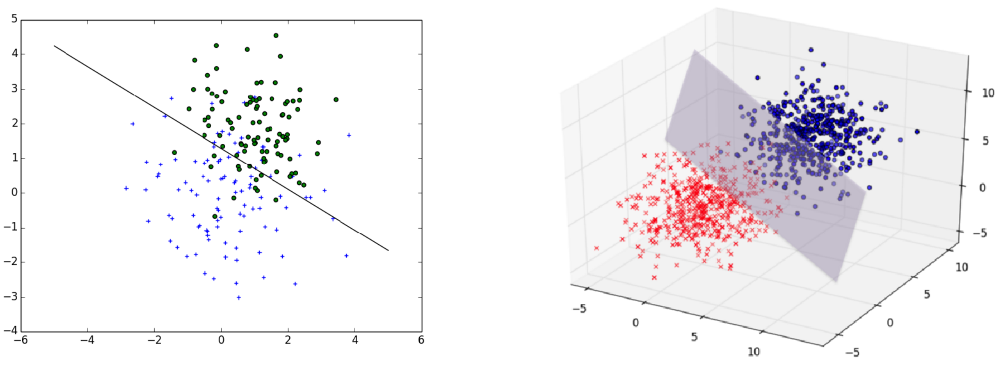
2.4 Linear Models for Classification

To minimize misclassification rate mentioned in 2.3.2, it is first necessary to be able to categorize the inputs. Linear models for classification use a *decision boundary* to decide on the outcome of classification. For linear models, the decision boundary is a linear function of the input vector $\vec{x}$ [1].

These models models use a *linear combination* of inputs, that is then transformed using a non-linear function. This linear combination is used in linear regression, which outputs a continuous variable. For the purposes of classification, it is necessary to take this variable and get a discrete output. In the case of linear models, it is done using the non-linear *activation function.*

2.4.1 Decision Boundary

In order to place an input vector $\vec{x}$ into one of the classes C_n, there needs to be a distinct boundary in the feature vector space that $\vec{x}$ belongs to. This is called decision boundary, it is the divide that separates regions in vector space belonging to individual classes. In terms of binary classification decision boundary is the dividing line that separates plane into two regions one for each class C_0, C_1. Decision boundaries in the case of linear classification models are defined by hyperplanes within the D-dimensional input space.



(a) Hyperplane in 2D feature space.

(b) Hyperplane in 3D feature space.

Figure 2.3: Examples of decision boundaries for 2D and 3D feature space.

Hyperplane is a subspace of the D-dimensional, therefore it is always a $(D-1)$-dimensional space. In the case of binary classification of 2D data, it would be a single line within the 2-dimensional plane.

2.4.2 Linear Regression and Linear Combination of Inputs

Linear model for regression, often referred to simply as linear regression, is commonly used in machine learning and statistics. This is not a model for classification, but the principle of linear combination of inputs is used in classification models. Linear regression establishes a relationship between input variables and the response variable y. As the name suggests, in its base form it models linear relationships between the variables.

Take for example an input vector $\vec{x}$ consisting of N input variables.

$$\vec{x} = (x_1, x_2, x_3, ..., x_n) \tag{2.7}$$

Each of these input variables is associated with a weight variable that is be altered throughout the learning process. These weights basically dictate how much of an impact does given input variable x_n have on the value of response variable y.

$$\vec{w} = (w_1, w_2, w_3, ..., w_n) \tag{2.8}$$

A simple way of calculating response variable is by linear combinations of these two vectors, with the addition of bias variable. that allows to take fixed offsets into account.

$$y(\vec{x}, \vec{w}) = w_0 + w_1x_1 + w_2x_2 + ... + w_nx_n \tag{2.9}$$

This, as stated previously, models a linear relationship. Linear regression can be made to model non-linear relationships by replacing $\vec{x}$ with some non-linear function of the inputs, $\phi(\vec{x})$ [9]. This is also known as the *basis function expansion*, in a simple form it could be a polynomial of the input variables $\phi(x) = [x_1, {x_2}^2, ..., {x_n}^n]$. Giving the following

$$y(\vec{x}, \vec{w}) = w_0 + \sum_{n=1}^{N} w_n\phi(x) = \vec{w}^T\phi_x + w_0. \tag{2.10}$$

What is often done to simplify the notation, as well as for the sake of convenience when doing calculations, is to expand the vector of variables adding 1 to it, $\vec{x} = (1, x_1, ..., x_n)$ and adding w_0 to the vector of weights $\vec{w} = (w_0, w_1, ..., w_n)$. Leaving us with

$$y(\vec{x}, \vec{w}) = \vec{w}^T\phi_{\vec{x}}. \tag{2.11}$$

Changing the basis function doesn't change the fact that this is still linear regression, the model remains linear in its parameters $\vec{w}$.

The output of this model is a continuous value of y based on the input vector $\vec{x}$.

2.4.3 Activation Function

Also called link function in statistics, is a function $f(\cdot)$ acting on the linear combination of inputs. This is generalization of the linear regression model and it is for this reason that these models are called *generalized linear models*.

Using a non-linear activation function with linear models doesn't make these models non-linear. The decision boundaries correspond to $f(\vec{w}^T\vec{x}) = constant$, so that $\vec{w}^T\vec{x} = constant$ and hence the decision boundaries are linear functions of x, even if the function $f(\cdot)$ is non-linear [1].

An example of a simple activation function can be a step function, that outputs 1 if the response variable is greater than zero and outputs 0 otherwise.

$$y(x) = \begin{cases} 1 & \vec{w}^T\vec{x} \geq 0 \\ 0 & else \end{cases} \tag{2.12}$$

Chapter 3

Logistic Regression as a Classification Model

Logistic regression is a machine learning model used to solve problems of classification. The name suggests, that it might belong to the category of models used for regression. It is called that, because it is used to estimate the probability of class membership. However, in classification, it is coupled with a decision rule that outputs a categorical variable based on the estimated probability.

Logistic regression models the posterior probability of $p(C_n|\vec{x})$ directly. Therefore it is a probabilistic discriminative model, these were mentioned in Subsection 2.3.1. The advantage these models have over probabilistic generative models is skipping the intermediate step of modeling class conditional densities $p(\vec{x}|C_n)$. Leaving aside computational issues and matters such as handling missing data, the prevailing consensus seems to be that discriminative classifiers are almost always to be preferred to generative ones [11].

Another thing to note is that in logistic regression, the number of adjustable parameters N is linearly dependent on the number of features. This makes it advantageous to use in classification problems with large number of features.

3.1 Activation Function in Logistic Regression

The approach to solving the problem of classification in logistic regression is to cast the problem in the form of generalized linear regression model. Logistic regression assumes that the log-odds can be expressed as a linear function of the input vector $\vec{x}$. Odds are defined as the probability of an event occurring over the probability of the event not occurring. Log-odds are simply the natural logarithm of the odds ratio.

$$\begin{gathered} y = \vec{w}^T\vec{x} \\ \log\frac{p(C_n|\vec{x})}{1-p(C_n|\vec{x})} = y \end{gathered} \tag{3.1}$$

The activation function used in logistic regression is derived from this relationship in the following way. Firstly, take the natural exponential function on both sides of the equation.

$$\frac{p(C_n|\vec{x})}{1-p(C_n|\vec{x})} = e^y \tag{3.2}$$

Then adding 1 on both sides and changing the subject from e^y to $1 - p(C_n|\vec{x})$.

$$\begin{aligned} \frac{p(C_n|\vec{x})}{1 - p(C_n|\vec{x})} + 1 &= e^y + 1 \\ \frac{1}{1 - p(C_n|\vec{x})} &= e^y + 1 \\ 1 - p(C_n|\vec{x}) &= \frac{1}{e^y + 1} \end{aligned} \tag{3.3}$$

Subsequently subtracting 1 and multiplying by -1 on both sides.

$$\begin{aligned} -p(C_n|\vec{x}) &= \frac{1}{e^y + 1} - 1 \\ p(C_n|\vec{x}) &= 1 - \frac{1}{e^y + 1} \\ p(C_n|\vec{x}) &= \frac{e^y}{e^y + 1} \end{aligned} \tag{3.4}$$

And lastly, dividing the numerator and denominator of the fraction by e^y to get the form of logistic sigmoid $\sigma(y)$. Logistic sigmoid plots the entire y-axis of the linear function to an interval of $[0, 1]$. This type of function is sometimes also called a 'squashing function' because it maps the whole real axis into a finite interval[1].

$$\begin{aligned} p(C_n|\vec{x}) &= \frac{\frac{e^y}{e^y}}{\frac{e^y+1}{e^y}} \\ p(C_n|\vec{x}) &= \sigma(y) = \frac{1}{1 + e^{-y}} \end{aligned} \tag{3.5}$$

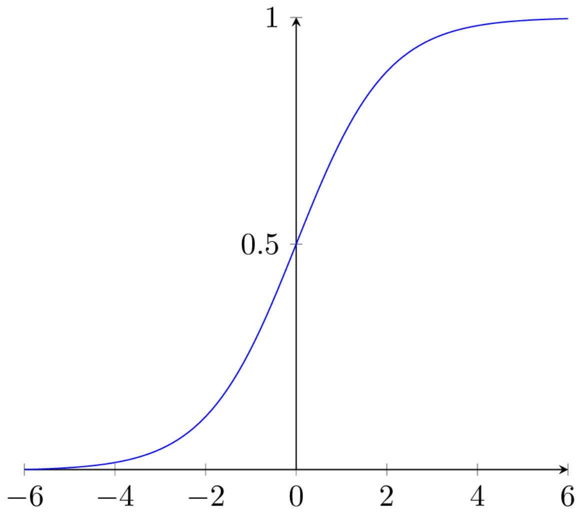


Figure 3.1: Activation function of logistic regression - logistic sigmoid.

Logistic sigmoid has the symmetric property $\sigma(-a) = 1 - \sigma(a)$, as can be seen in 3.1. Because of this symmetry the probability of $p(C_1|\vec{x}) = 1 - p(C_0|\vec{x})$.

3.2 Cross Entropy Error Function

In logistic regression the *cross entropy* error function is used to assert performance of the model. This function is the negative logarithm of the likelihood function. The reasons behind minimizing error, as opposed to maximizing likelihood are outlined in Subsection 2.3.2. To get the likelihood function it is first necessary to take the derivative of the logistic sigmoid function.

$$\begin{aligned}\sigma(a) &= \frac{e^a}{1+e^a}\\ &= e^a(1+e^a)^{-1}\\ \sigma'(a) &= e^a(1+e^a)^{-1} + e^a(-1)(1+e^a)^{-2}e^a\\ &= \frac{e^a(1+e^a)}{(1+e^a)^2} - \frac{e^{2a}}{(1+e^a)^2}\\ &= \frac{e^a}{(1+e^a)^2}\\ &= \frac{e^a}{1+e^a}\cdot\frac{1}{1+e^a}\\ &= \sigma(a)(1-\sigma(a))\end{aligned} \tag{3.6}$$

As can be seen in 3.6, the derivative of this function can be expressed in terms of logistic sigmoid itself. The likelihood function expresses how probable the observed dataset is for different settings of the parameter vector $\vec{w}$ [1]. The likelihood function is therefore the product of the predicted probabilities of the N individual observations in the observed dataset. In the case of binary classification, for a dataset $\vec{x}_n, t_n$, where $t_n \in 0, 1$ with $n = 1, ..., N$, the likelihood function can then be written as follows [1].

$$p(t|\vec{w}) = \prod_{n=1}^{N} y_n^{t_n}(1-y_n)^{1-t_n} \tag{3.7}$$

Here t_n is the correct class assignment and y_n is the value of response variable from logistic sigmoid for given data point n. In other words, y_n is the posterior probability $p(C_1|\vec{x}) = \sigma(\vec{w}^T\vec{x})$. Note that the model will always model only one of the probabilities $p(C_0|\vec{x})$ and $p(C_1|\vec{x})$. To get the cross entropy function, it is necessary to take the negative logarithm of this likelihood function.

$$E(\vec{w}) = -\sum_{n=1}^{N}\{t_n \log(y_n) + (1-t_n)\log(1-y_n)\} \tag{3.8}$$

The cross entropy function is convenient to use as an error function for two reasons.

Firstly, the value of this function is always non-negative $E(w) > 0$. This is because both of the logarithms inside the sum output a negative number,as the value of y_n can only be from the range of 0 to 1, and there is a negative sign before the sum.

Secondly, the function tends towards zero, making this a convex optimization problem. The reason for this is that in cases where to model is performing well the value of cross entropy is close to 0. Consider the case, when the correct label $t_n = 0$ and the response variable is $y_n \approx 0$. The first factor in the sum is 0 because of the class label $t_n = 0$ and the second factor is $\log(1-y_n) \approx \log(1) \approx 0$. Same thing applies when the second factor is 0 and $y_n \approx 1$.

3.3 Training the Model

The cross entropy error function is used when training the model to achieve optimal performance. The goal is to find the minimum of the error function. The analytical solution to this problem is impossible, therefore it is necessary to use numeric methods. The negative gradient $-\nabla E(\vec{w})$ is used to find the direction of the steepest descent. To get the gradient of the cross entropy function 3.8, take its partial derivative with respect to the weights.

$$\begin{aligned}\frac{\partial}{\partial \vec{w}} E(\vec{w}) &= -\sum_{n=1}^{N} (\frac{t_n}{y_n} - \frac{1-t_n}{1-y_n}) \frac{\partial}{\partial \vec{w}} y_n \\ &= -\sum_{n=1}^{N} (\frac{t_n}{y_n} - \frac{1-t_n}{1-y_n}) y_n(1-y_n) \frac{\partial}{\partial \vec{w}} \vec{w}^T \vec{x}_n \\ &= -\sum_{n=1}^{N} t_n(1-y_n) - (1-t_n) y_n \vec{x}_n \\ &= -\sum_{n=1}^{N} (t_n - y_n) \vec{x}_n\end{aligned} \tag{3.9}$$

Where t_n is the correct class label, y_n is the value of logistic sigmoid $\sigma(\vec{w}^T\vec{x})$ for current weights $\vec{w}$ and input vector $\vec{x}$. It was necessary to apply chain rule twice and use the derivative of logistic sigmoid from 3.6 to get this partial derivative. The gradient is then used when adjusting the vector of weights $\vec{w}$ to take a step of size η in the direction of steepest descent.

$$\begin{gathered}\nabla E(\vec{w}) = \sum_{n=1}^{N} (y_n - t_n)\vec{x}_n \\ \vec{w}^{(\tau+1)} = \vec{w}^{(\tau)} - \eta \nabla E(\vec{w}^{(\tau)})\end{gathered} \tag{3.10}$$

This technique is called *gradient descent*. It is an iterative algorithm, where the weights are adjusted repeatedly until the error function converges, or its value falls below a certain threshold. However, for a linearly separable dataset there is a multitude of equally valid solutions, because any of the hyperplanes separating both classes entirely will give rise to the same posterior probabilities [1]. The one that the algorithm converges on is entirely dependent on the initial parameters $\vec{w}$ and the step size η. The algorithm does not choose one equally good solution over the other.

Chapter 4

Design and Implementation of the Demo Application

This chapter documents the design and the implementation of the demo application. The aim of this application is to provide a complementary learning tool to the literature documenting this machine learning model. This application should be able to convey intricacies of classification in general and logistic regression in particular to the end user. The demo application along with the documentation should help visualize and conceptualize the workings of the algorithm, even to the users that may not be familiar with classification and machine learning.

Theoretical knowledge from the previous chapters is used to ascertain which important parts of the learning process need to be displayed and controlled by the user. Namely, the training dataset, decision boundary and model training, cross entropy error function plot and the gradient descent. The way these parts are implemented is talked about in the following sections.

4.1 Designing the UI for the Demo Application

The goal in designing the demonstrative application is to provide an interactive environment, where te end user can get familiar with the algorithms and explore the impact certain options have on the outcome of the classification. The relevant parts were separated into three groups.

Firstly, it is important to allow the user to view and input and edit data in order to generate a training dataset. This also creates the necessity for saving and loading of data so that the user may return to the previously created training datasets, or perhaps view a preset one. This data is generated and displayed on a separate chart along with the decision boundary of the model. It is necessary to allow the user to interact with the model and set its parameters to see the impact it has on the decision boundary. Similarly, it is important to allow the user to modify variables involved in the training process and allow him to step through the algorithm one step at a time.

The next needed functionality was displaying the cross entropy error function plot from Equation 3.8. This needed to be done in an understandable manner, as it is a function of 3 parameters. It was necessary to allow the user to interact with the plot, rotate it and change its scope. In addition to this, the path the gradient descent algorithm takes should be displayed on the same chart to show how the weights were altered throughout the learning process.

And finally, the last aspect of the logistic regression that I decided to show was how each data point is classified by the logistic sigmoid. Displaying the posterior probability $p(C_1|\vec{x})$ for each individual data point along with all the relevant variables and their values.

In the first iteration of the application I tried to display all of the relevant information on a single large dashboard. However, this proved to be far too unwieldy. The charts, tables and the user toolkit were far too cluttered. It was difficult to distinguish some things on smaller monitors and resolutions.

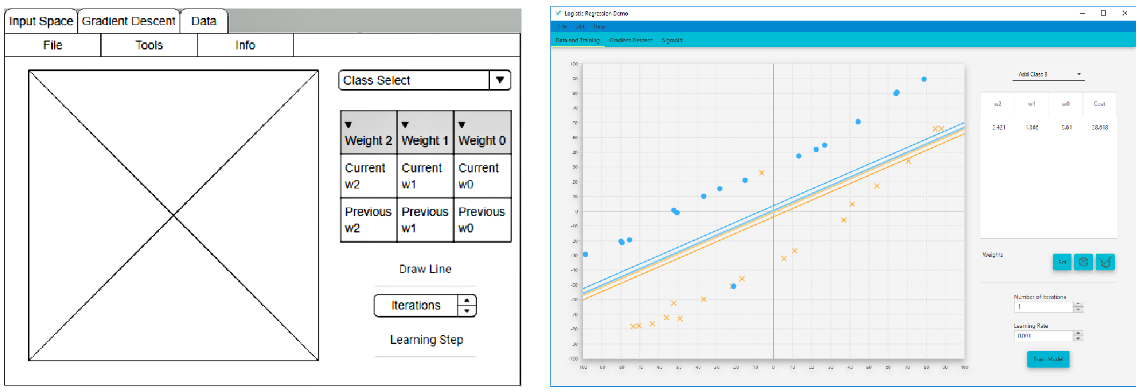


Figure 4.1: Initial mockup of the UI and subsequent implementation of that design.

The solution to this problem was to separate the application into three tabs 4.1, each containing a subset of all the necessary parts. This way the user can move them around and choose the way he wants to view them.

4.2 Implementing the Demo Application

The programming language of my choice was JavaFX [13]. The reasons behind using this language were mainly the portability and the appealing user interface design options. It allows for generation of separate FXML files which contain the UI elements and are separate from the logic of the application. There is also a very helpful open source tool for viewing and editing these files [14]. The application can be further styled by use of CSS files, which allow for creation of truly good looking interface.

The demo application is mainly focused on smaller datasets, which can be used to illustrate aspects of the model. Therefore the performance of the application should not come into question, as the amount of data processed will be relatively small.

4.2.1 Draggable and Detachable Tabs

The main part of the UI is a TabPane consisting of three tabs, each containing different relevant information about the machine learning model. These tabs are implemented using the open source FXTabs library [2], which I modified in order to to better suit the needs of this application.

This library allows for creation of new windows by dragging the tabs outside the scope of the TabPane by creating a new Stage. A separate Scene is created for the tabs, FXML file is loaded and set as the content for each of these tabs. This means that all of the information and the state of the UI persist through the movement and creation of the new windows. This functionality is similar to how the Internet browsers function and allows the user to customize their viewing experience.

The changes made to this library are changing the style of the windows that it created, allowing the use of components from the JFoenix library [6] and additional functionalities in movements of the tabs. The tabs can be navigated through using the arrow keys and the newly created tabs are returned to the main TabPane if they are closed.

4.2.2 Displaying the DataSet and the Model

The first tab in the TabPane is the chart that displays the dataset, decision boundary and the posteriori. This display is done on a modified LineChart from the JavaFX chart library. The CSS styling of the data points disables connection of these separate points creating a part ScatterPlot part LineChart.

Additional logic is added for drawing of straight lines to be able to display decision boundaries and posterior probabilities. The lines are drawn by finding their intersection points with the chart, if the line is beyond the scope of the chart none are found and the line is not drawn. Several lines are drawn at certain thresholds of the posterior probability. Nine in total displaying the range of $p(C_1|\vec{x})$ from 0 to 1 in 0.125 increments creating a total of 9 lines. The way the linear functions for these posteriori are found is first taking the decision boundary, line for which $p(C_1|\vec{x}) = 0.5$, which is known from training the model. Then getting the slope k of a perpendicular line to the decision boundary by taking the negative reciprocal of the decision boundary slope. Subsequently, solving the logistic sigmoid in reverse for each value of $p(C_1|\vec{x})$ to get the slope intersect form of their function.

4.2.3 Training of the Model

The classifier is implemented using the mathematical formulae from Chapter 3. The UI allows the user to set the weights $w0, w1, w2$ for the classifier. It also allows for setting of the learning rate and the number of iterations the algorithm should calculate. The user can step through one step at a time. The history of steps taken is displayed in a TreeTableView alongside the chart along with the respective cost function values for given parameters. The user can choose to return to a previous state of the classifier by choosing a row from this TreeTableView.

4.2.4 Plotting of the Cost Function

The plotting of the cost function proved to be a more difficult problem than was previously anticipated. The cross entropy error function that needs to be plotted is a function of 3 parameters $f(w_0, w_1, w_2)$. The problem is that to visualize this $4D$ data there needs to be a cost function value associated with every point in the $3D$ vector space of parameters. This is because each vector represents a decision line on the chart from 4.2.2.

There are multiple techniques available to display this sort of data. Most prominent of them are scalar field, scatter plot matrix, slice contour plot and a $3D$ scatter plot with $4D$ data. I settled upon the $3D$ scatter plot with the value of cost function being represented by color. This approach proved to be a lot more clear and understandable than scalar fields or scatter plot matrices. At the same time it provided information about the entire interval of the cost function that is being displayed, unlike the contour plot which displays 3 perpendicular planes that the user moves around.

It was necessary to threshold the values of the posterior probabilities output by the logistic sigmoid. Because of the floating point precision in some cases due to rounding the algorithm was calculating the logarithm of 0 or approximately 0, which is $-\infty$. This made the color scaling on the cost function plot incredibly unbalanced. After some deliberation and testing I settled on values 1.10^{-15} and $1-1.10^{-15}$, after which there was no perceptible change in the plot.

This scatter plot was implemented using the JZY3D library for scientific plotting [7]. The controls implemented allow the user to choose which interval of the cost function plot he wishes to see. The library covers all of the rotations and translations of the plot to allow the user to view it from all angles. This library also allows for continuous rotation of the scatter plot by double-clicking on the plot. This comes in very useful when visualizing $3D$ data.

4.2.5 Plotting the Logistic Sigmoid

The last tab displays the plot of the logistic sigmoid. The function is plotted using the JavaFX charts and the data points on the chart are displayed in the same fashion as in the Subsection 4.2.2. It is here to show the user how the each data point is classified by the model. This allow the user to view all of the misclassified data points and the posterior probability $p(C_1|\vec{x})$ they are associated with.

Below the plot of the logistic sigmoid function are two tables created using the JFoenix library. In tables are all of the data points, they are separated by class. Each table contains the label of a data point, its coordinates, value of the linear combination of parameters and value of the logistic sigmoid function for that data point. Undesirable data points can be removed from the dataset by bringing up context menu and choosing Delete option.

Chapter 5

Findings about the Plot of the Cross Entropy Error Function

During the development of the application I came to the realization that the plot of the cross entropy error function is not what would be usually expected. There is little documentation available about the visualization of this function. In Section 3.2 is the proof of convexity of the cross entropy function. If you consider how a convex function looks in $2D$ and $3D$ vector space you can begin to make assumptions about the plot of the cost function of linear logistic regression.

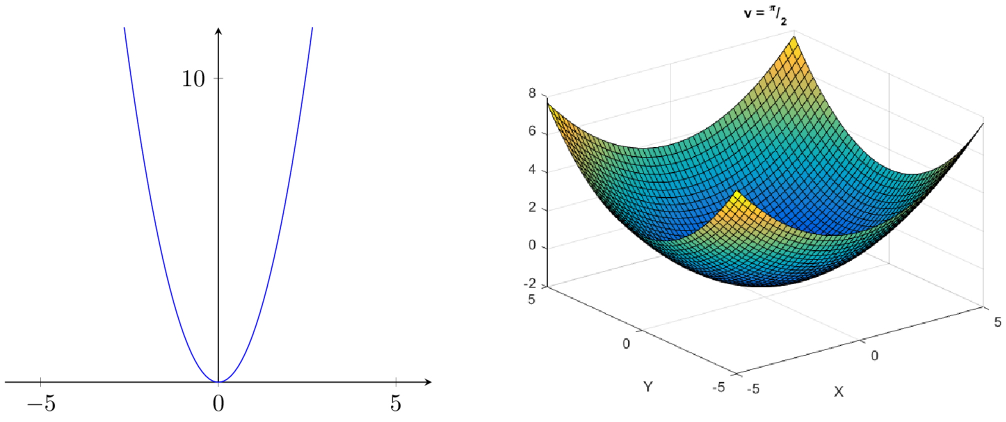
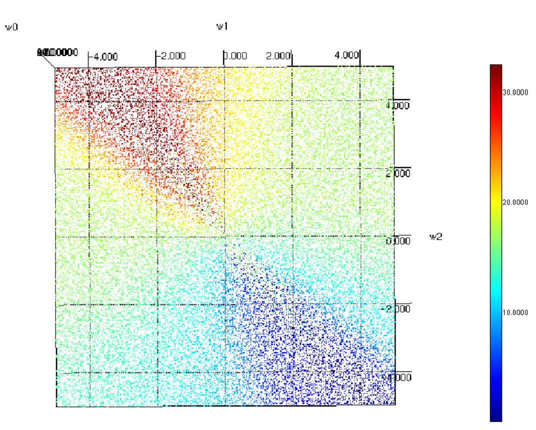


Figure 5.1: Plots of simple convex functions in 2D and 3D vector space.

Considering how a simple convex functions of 1 and 2 parameters look 5.1, the assumptions I made were that the cost function would take the form of a concentric spheres, or spheroids. These spheroids would be centered around the minimum of the cross entropy function and the gradient descent algorithm would slowly converge towards this center.

What was not taken into consideration when making these assumptions is that there are infinitely many equally valid solutions for a given classification problem. You can have multiple weights in $\sigma(w_0, w_1, w_2)$ that represent the same linear equation $w_2y+w_1x+w_0 = 0$ due the linear dependence of their equations. This creates a region in the $3D$ vector space where the value of the cross entropy function is the same.


(a) Horizontal slice of the function with w0 from interval (5,5)

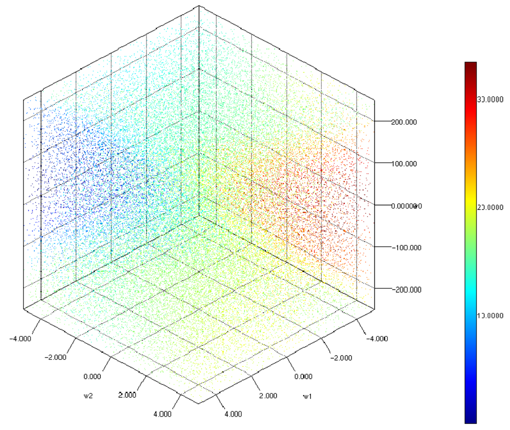
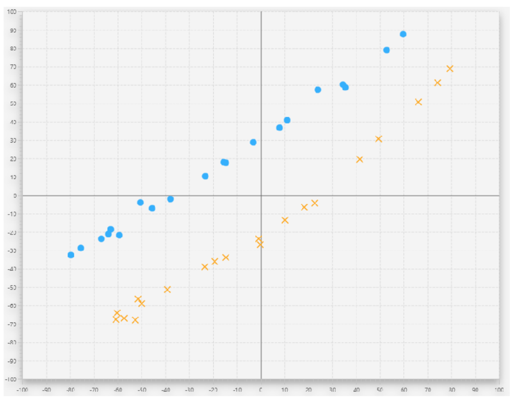

(c) Rest of the functions plot

Figure 5.2: Plots of the cross entropy error function of the linear logistic regression model for a linearly separable dataset.

The two figures in 5.2 are taken from the demonstrative application. These cost functions were calculated for a simple linearly separable dataset (a) from 5.3. In the figures the valid or good solutions for which the cost function is low are depicted in deep blue, likewise incredibly bad solutions are depicted in deep red. All the other hues of color represent values somewhere in between.


(a) Simple linearly separable dataset

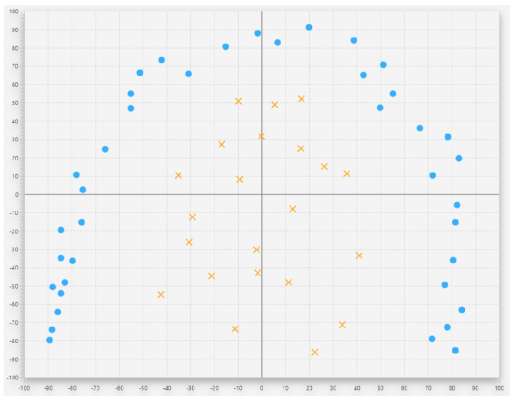
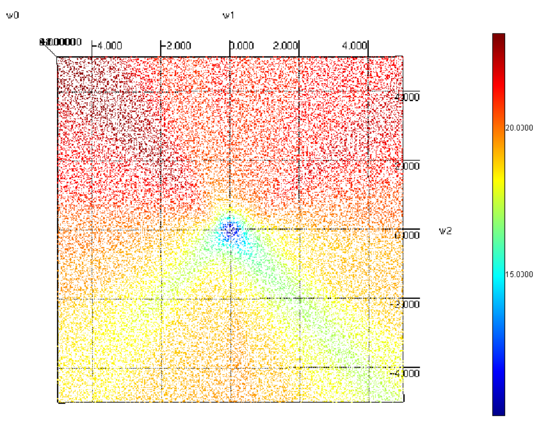

(b) Dataset separable by a polynomial

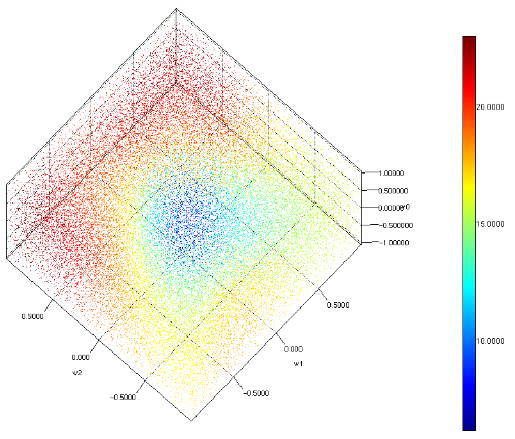
Figure 5.3: Examples of plots of cross entropy error functions are provided for these two datasets.

This cross entropy function has a very symmetrical shape. This is because for every good solution that separates both classes almost entirely and classifies them mostly correctly, there exists the same decision boundary with inverted posterior probabilities that misclassifies all of these data points. Mostly all of the linearly separable datasets have a cost function that looks like two conical shapes opposed one another, with one region of the function with low cost function values and one with equally high cost function values.

Another interesting cost function is of a dataset that could be potentially separable by a polynomial logistic regression. The linear logistic regression cannot reasonably separate these two classes in the dataset (b) from 5.3. Resulting into a number of solutions that are mediocre. This creates a sort of cross section of regions with similar values of the cross entropy function centered around the point $(0, 0, 0)$. This center of relatively low error given the circumstances contains models, which predict probability of $p(C_1|\vec{x}) \approx 0.5$ for most of the data points. This pattern in particular is much easier to observe in person when using the demo application by using the automatic rotation functionality that the scatter plot provides. The deep red regions with large cost function values can be seen in higher contrast.



(a) Horizontal slice of the function with w0 from the interval (-5,5)



(c) Central section of the plot

Figure 5.4: Plots of the cross entropy error function of the linear logistic regression model for a dataset that is not linearly separable.

These two datasets are the only ones I mention here. For those that are interested in additional datasets and their corresponding cross entropy functions, there are some stored on the CD provided in the appendix A that can be viewed in the demonstrative application.

Chapter 6

Conclusion

The goal of this thesis was to study the inner workings of the machine learning model linear logistic regression. Subsequently, use the obtained information to create a demonstrative application that would showcase aspects of this model. As well as, document the way this model functions and all the mathematics involved in a clear and understandable manner.

The first step was the study of pattern recognition and classification in general, in order to gain an understanding of given problematic. Next, it was necessary to understand the categories of generalized linear models and probabilistic discriminative models and how they differ from other models used for classification. Followed by the derivation and documentation of the relevant mathematical formulae necessary for logistic regression.

With this understanding of the model, a demonstrative application was designed and implemented in a way that illustrates how the the dataset and functions used relate to each other. This application allows the user to manipulate parameters of the machine learning model and inspect the effects these changes have on the outcome of the classification.

Going forward, if I were to expand on this work I would start from the ground up and use the already developed demo as a part of a larger application. In my personal opinion, in order to gain a complete understanding of this model, it is necessary to start at the beginning. The best thing to do would be to create an application which would consist of several smaller demos each covering their subject in depth. This application would start with the linear regression model. From there, move on to a simple model that uses a discriminant function, like the perceptron. Then cover the linear logistic regression and finish with the polynomial and multinomial logistic regression. This way it would be possible to show how these models build on top of one another, the relationship between these models and why it all works this way.

Another addition that could be made is an application that would generate datasets which could be used in the demonstrative application. There would be a choice from a number of distributions to generate different types of data. This application could also analyze the data that were created in the demonstrative application which user found had interesting effects on the machine learning model.

The findings about the plot of the cross entropy error function are also worth analyzing further. The interesting forms that the function takes depending on the training dataset are not what was expected. This could warrant a further study of this model and its error function.

Bibliography

[1] Bishop, C. M.: *Pattern recognition and machine learning*. springer. 2007. ISBN 978-0387-31073-2.

[2] FXTabs: Draggable and detachable tabs in JavaFX 2. https://berry120.blogspot.cz/2014/01/draggable-and-detachable-tabs-in-javafx.html.

[3] Gantz, J.; Reinsel, D.: The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. *IDC iView: IDC Analyze the future*. vol. 2007, no. 2012. 2012: pp. 1–16.

[4] Icons8: Icons Repository . https://icons8.com/.

[5] Jaynes, E. T.: *Probability theory: the logic of science*. Cambridge university press. 2003.

[6] JFoenix: JavaFx Material Design Library . http://www.jfoenix.com/.

[7] Jzy3d: Scientific 3-D plotting software. Software and documentation . http://www.jzy3d.org/index.php.

[8] Mitchell, T. M.: *Machine Learning*. McGraw Hill. 1997.

[9] Murphy, K. P.: *Machine learning: a probabilistic perspective*. MIT press. 2012.

[10] Ng, A.: CS229 Lecture notes. *CS229 Lecture notes*. 2000.

[11] Ng, A. Y.; Jordan, M. I.: On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Advances in neural information processing systems*. 2002.

[12] Nielsen, M. A.: *Neural networks and deep learning*. Determination Press. 2015.

[13] Oracle: Java Platform, Standard Edition (Java SE) 8. https://docs.oracle.com/javase/8/javase-clienttechnologies.htm.

[14] Scene Builder: A Visual Layout Tool for JavaFX Applications. http://gluonhq.com/products/scene-builder/.

[15] Turner, V.; Gantz, J. F.; Reinsel, D.; et al.: The digital universe of opportunities: Rich data and the increasing value of the internet of things. *IDC Analyze the Future*. 2014: page 5.

[16] Weaver, J.; Gao, W.; Chin, S.; et al.: *Pro JavaFX 8: A Definitive Guide to Building Desktop, Mobile, and Embedded Java Clients*. Apress. 2014.

[17] Zumel, N.: The Simpler Derivation of Logistic Regression. 2011.
Retrieved from: https://web.archive.org/web/20170427082928/http://www.win-vector.com/blog/2011/09/the-simpler-derivation-of-logistic-regression/

Appendix A

Contents of the CD

- LaTeX source code for this document
- PDF of this document
- LaTeX source code for the poster
- PDF of the poster
- Source code of the application
- Generated documentation for the application source code
- Executable file of the application
- Pre-generated data sets